Real-time Global Illumination using Irradiance Probes

Simon Sedlacek*
Supervised by: Jiri Bittner†

Department of Computer Graphics and Interaction
Czech Technical University
Prague / Czech Republic

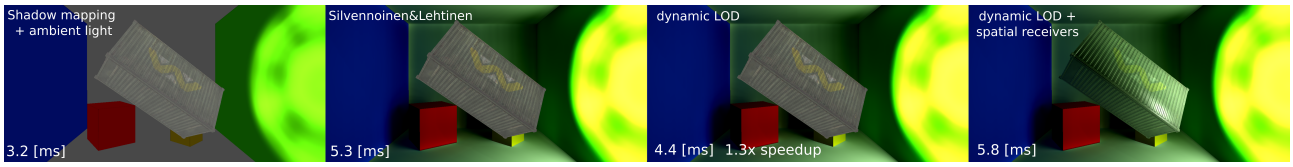

Figure 1: Reflector on the right indirectly illuminates the scene. Shadow mapping + ambient light: no indirect illumination. The reference method proposed by Silvennoinen and Lehtinen [4]: global illumination only for a static scene. dynamic LOD: significant speedup of the proposed method with an only small error in the rendered image. dynamic LOD + spatial receivers: global illumination with the support of shading for dynamic objects.

Abstract

I present two extensions to a recently introduced method for real-time global illumination based on sparse irradiance caching. The first extension allows the computation of global illumination for dynamic objects with the support of various shading techniques (normal mapping, specular reflections). The second extension uses a dynamic level of detail for global illumination, which is able to significantly improve performance without noticeable errors in the rendered image.

Keywords: real-time global illumination, precomputed radiance transfer, light propagation, spherical harmonics, dynamic level of detail, performance improvement

1 Introduction

Calculation of a correct light propagation through a scene in computer graphics is one of the most impactful effects there is. Accurate light transport is often the effect, which blurs a line between computer graphics and reality the most. As impactful this effect is, it is also costly. To compute such an enormous task of light propagation in real-time even more problems show up. A standard way how to solve this problem today is to calculate only a single light path: from a light source to a first hit point. The rest of the light propagation is then approximated with ambient light.

As it is costly to compute the light path for every photon in real-time, there was an idea to precompute these paths and reuse them in real-time rendering. The class of algorithms, which precompute lightpaths and reuse them in real-time, is called Precomputed Radiance Transfer (PRT).

For a starting point of my work, I chose a newly released algorithm from the PRT class: Real-time Global Illumination by Precomputed Local Reconstruction from Sparse Radiance Probes by Silvennoinen and Lehtinen [4]. This algorithm allows to precompute light paths and due to re-rendering of computed light paths, it is able to approximate global illumination in real-time for dynamic lights. As for every method, there are some drawbacks. This method does not support dynamic objects and effects like specular reflections are costly.

In this work, I present two extensions which solve the drawbacks of the algorithm and push it towards real-life usage in commercial rendering engines. The first extension is able to shade dynamic objects in real-time with global illumination using a sparse voxel grid. This extension is also capable of normal mapping and specular reflections approximation without the need for costly radiance transport. The second extension uses a dynamic Level of Detail to omit computations and improve performance.

2 Related Work

Simulation of light propagation can be achieved in real-time using two main approaches: view-dependent or view-independent methods.

View-dependent methods use an idea of tracing rays from a camera view. These methods provide fully dynamic scenes and support even high-frequency spatially-varying BRDF (such as specular reflections, mirrors, etc.). However, the computation of visibility in real-time is costly.

Thus it is usually possible to compute only a few light bounces. Due to possible insufficient sampling of rays, these methods exhibit noise artifacts. View-dependent methods can either be pure ray-tracing methods or an approximation of a light bounce using voxel space such as Voxel Cone Tracing [5].

View-independent methods have to compute light reflectance independently of the camera position. If we omit dynamic scenes and focus purely on static scenes a lot of information about light propagation can be precomputed. The class of algorithms using this idea is called Precomputed Radiance Transfer. For storage of light paths and information about light reflectance Spherical Harmonics are used. Green [2] describes the basic usage of Spherical Harmonics in computer graphics.

Spherical Harmonics are good delivering light intermedian on low-frequency surfaces, such as diffuse surfaces. For example, Sloan et al. [6] used Spherical Harmonics to capture how the surface of a geometry reacts to an incoming radiance. They stored this information on surface points and were able to reconstruct shading with self-shadowing not only for polygon meshes but also for volumetric geometry such as clouds and fog.

To support fully dynamic lights Silvennoinen and Lehtinen [4] decomposed the light transport to two major stages. They separated light gathering (using radiance probes) and light propagation (using radiance receivers). Furthermore, with the usage of Singular Value Decomposition (SVD), the truncation of transport matrices leads to memory and time efficient algorithm for real-time global illumination.

3 The proposed method

My work is based on the algorithm proposed by Silvennoinen and Lehtinen [4]. Since my extensions use this method as a basis, I describe this method first.

Computing interaction of every light sample point with every other is demanding on time. To solve this problem, the authors factorized irradiance transport into two stages. The method uses probes and receivers for transport of irradiance.

The probe is a point in space in which incoming irradiance is measured from many directions (ideally it would be from all directions). It stores this measured irradiance, which is then used by receivers. The set of probes is sparse (the usual density is around 0.02 probes per m^3).

The receiver is a point on the surface of the scene. It uses surrounding probes to fetch irradiance. For each probe, the receiver fetches the irradiance from a correct direction. The set of receivers is dense (the usual density is around 80 receivers per m^3).In the implementation, receivers are texels of a lightmap. The lightmap is then used when meshes of the scene are rendered.

An example of how this works is shown in Figure 2. Probes describe irradiance from many directions. Receivers use the irradiance from probes to illuminate the lightmap.

During the second cycle, the lightmap has been illuminated from the first cycle. Thus probes will accumulate irradiance not only from the light source but also from the lightmap. This cycling of irradiance through the system results in global illumination.

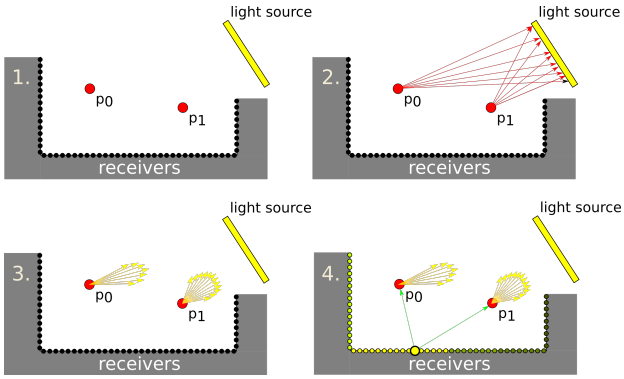

Figure 2: The rendering cycle. The first image: two probes (p_0 and p_1) in space visualized as red dots, many receivers on scene surface as black dots and a single light source visualized as a yellow rectangle. The second image: both probes measure irradiance in the scene. The third image: probes describe irradiance from many directions. The fourth image: receivers fetch irradiance from probes and accumulate it. Irradiance measured in receivers is then used to illuminate scene meshes.

3.1 Spherical Harmonics

The method uses Spherical Harmonics to transport irradiance.

Spherical Harmonics (SH) are functions, which allow approximation of any k-dimensional function that is defined on a surface of a sphere. Naturally, SH allow reconstruction of the stored signal in any direction.

An example of Spherical Harmonics usage in computer graphics has been described by Green [2]. SH basis functions are commonly noted as $Y_l^m(\vec{\omega})$, where $\vec{\omega}$ is parametrized position on a unit sphere, l is a degree of SH and m is an index of a function in this degree [6].

A signal of encoded function can be obtained by the weighted sum of several basis functions:

$$f(\vec{\omega}) \approx \sum_{l=0}^{n-1} \sum_{m=-l}^{l} \lambda_l^m Y_l^m(\vec{\omega}) \qquad (1)$$

where n is the maximum degree of SH (sometimes called a band of SH).

The numbers λ_l^m are called SH coefficients. SH coefficients encode the function itself and are measured from the samples to obtain $f(\vec{\omega})$. For simplification of Equation 1, I am going to use a notation from [4]: $\lambda_l^m = \lambda_{ij}$, where i is an index of some particular measured function and j is

an index of a basis function obtained as $j = l^2 + l + m$. To store the signal using SH we must obtain SH coefficients λ_{ij}. The equation is as simple as final signal reconstruction the process is just reversed:

$$\lambda_{ij} \approx \frac{4\pi}{N} \sum_{i=1}^{N} f(\vec{x}_i) Y_j(\vec{x}_i) \tag{2}$$

where x_i is a sample from a set of sampled irradiance. Equation 2 uses Monte-Carlo integration with the weight of every sample equal to 4π (samples on the surface of a sphere). Once SH coefficients are known the reconstruction of the signal can be computed by Equation 1.

Every probe has its unique set of measured λ_{ij} SH coefficients. These coefficients are then provided to receivers.

The degree of SH crucially affects the accuracy of the reconstructed signal. The higher the degree is the better approximation is provided (see Figure 3).

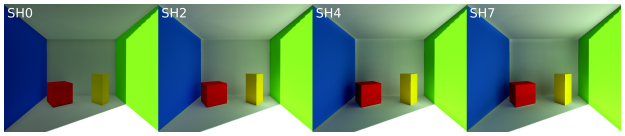

Figure 3: The degree of Spherical Harmonics affects the accuracy of the signal reconstruction (degrees 0,2,4,7 shown).

3.2 Per Receiver Irradiance Gathering

The irradiance in a receiver is computed by gathering irradiance from a probe (using its λ_{ij} SH coefficients). Receiver x gathers irradiance from probe p_i in direction ω if both the probe and the receiver see in that direction exactly the same surface.

To store information about this visibility match for each receiver x and probe p_i the authors introduced the following visibility function:

$$K_{ij}(x,\omega) = \frac{w_i(x) V_i(\omega) Y_j(\Psi(\omega))}{\sum_k w_k(x) V_k(\omega)} \tag{3}$$

where j is an index of an SH basis function, k are indexes of all probes, $w_i(x)$ is a distance weight function between the probe p_i and the receiver x [4]. ω is a direction from the receiver to a surface visible to the receiver. $\Psi(\omega)$ is a direction from the probe to the same surface. $V_i(\omega)$ is a binary function. $V_i(\omega) = 1$, if the surface visible by the receiver x from the direction ω, is also visible to the probe p_i. Function $K_{ij}(x,\omega)$ encodes visibility match between probe p_i and receiver x weighted by their distance. This function can be then used to compute a visibility match from any direction ω. The visibility match is denoted as α_{ij} and can be computed as an integral using $K_{ij}(x,\omega)$ for all ω [4]:

$$\alpha_{ij} = \int_{\omega \in \Omega} K_{ij}(\omega) d\omega \tag{4}$$

Gathered irradiance in some receiver x is computed in real-time using the following equation:

$$I(x) \approx \int_{\omega \in \Omega} \sum_{ij} \lambda_{ij} K_{ij}(\omega) d\omega = \sum_{ij} \lambda_{ij} \alpha_{ij} \tag{5}$$

where i is an index of a probe. Notice similarities with Equation 1. α_{ij} can be seen as $\forall \omega : Y_j(\omega)$ enriched with weighted visibility. Values α_{ij} are precomputed for all receivers and stored to be used in real-time rendering.

3.3 Gathering Acceleration

Because the number of receiver coefficients α_{ij} is huge, the authors factorized irradiance transport using Clustered Principal Component Analysis (CPCA). Receivers are divided into clusters (kD tree division) in the precomputation part. Coefficients α_{ij} of receivers in a certain cluster are put vertically into transport matrices (matrix A_c for cluster c in Equation 6) and SVD matrices are computed.

$$A_c = U_c \Sigma_c V_c^T \tag{6}$$

This decomposition is truncated (leaving only the 32 biggest singular values from matrix Σ_c) to ease memory requirements [4]. If the SVD projection was not sufficiently accurate, the cluster is recursively split into half.

The algorithm is divided into precomputation part and real-time rendering. During the precomputation part, all visibility coefficients α_{ij} are computed using ray-casting and stored in truncated SVD matrices (U_c and $\Sigma_c V_c^T$). Also, all hitpoints for all probes (set of samples x_i from eq. 2) and values of basis functions $Y_j(\vec{x}_i)$ are precomputed and stored.

During real-time rendering, all probes compute their λ_{ij} SH coefficients using precomputed hitpoints and values of basis functions. Receivers then compute their irradiance using λ_{ij} coefficients and precomputed SVD matrices. Dynamic light sources can be implemented using standard shadow mapping to illuminate the lightmap.

3.4 Drawbacks

The proposed method [4] does not support dynamic objects and effects like normal mapping or specular reflections are possible only if radiance is transported. As it was already noted by the authors, transport of radiance is noticeably more demanding on both memory and computation time due to the higher dimension of transport in every stage. Also, the method does not scale well with increasing number of receivers, as it has to recompute the whole lightmap in every frame.

I have created two extensions which aimed to solve these drawbacks. The first extension (section 4) is able to shade dynamic objects with global illumination and supports effects like normal mapping or specular reflections. The second extension (section 5) uses a dynamic Level of Detail to omit computations and improve performance.

Both of the extensions are described in the following chapters.

4 Shading of Dynamic Objects

In this section, I present my first extension, which is capable of shading dynamic objects with global illumination using a sparse voxel grid. To measure irradiance on a surface of dynamic objects, I used a special version of irradiance receivers. I call them spatial receivers. Similarly to The Irradiance Volume [3], these receivers measure irradiance from certain directions. Particularly in such directions to be able to estimate irradiance in all directions. In my implementation, a single spatial receiver is measuring irradiance from 6 different directions, which are equal to axes of the scene ($\pm x$, $\pm y$ and $\pm z$). However, the method is not constrained to a fixed number of directions. More directions can provide a better approximation of irradiance.

In every direction, I measure irradiance with a weight equal to a normalized degree between the direction of incoming irradiance and the current direction vector. The weight is computed with the following equation:

$$w(d,I) = \frac{\frac{\pi}{2} - acos(max(dot(d,I),0)}{\frac{\pi}{2}} \tag{7}$$

This equation gives us the ability to combine irradiance from different directions into a single value. In the case of spatial receivers, I chose to cluster every direction separately as it was more coherent.

I voxelized the scene with a regular grid and placed a single spatial receiver in every vertex of the grid, as it was proposed by Greger et al. [3]. The algorithm takes an approximate number of desired spatial receivers as an input from the user. I estimate the number of spatial receivers in every axis using the method of Amanatides et al. [1].

$$N_z = \left\lceil \sqrt[3]{\frac{Nz^2}{xy}} \right\rceil, N_y = \left\lceil \sqrt{\frac{Ny}{N_z x}} \right\rceil, N_x = \left\lceil \frac{N}{N_y N_z} \right\rceil \tag{8}$$

where N_z, N_y, N_x is a number of cells in z-, y-, x-axis. N is a desired number of cells and $[x,y,z]$ is a size of the scenes axis-aligned bounding box (AABB).

To extend this idea, in my implementation every dynamic object has its AABB. This AABB is tested against the irradiance grid, so I can detect, which spatial receivers have to be computed. The rest does not compute its irradiance at all.

It can happen and it will happen that some spatial receivers will be placed inside objects of the static scene. These receivers would not receive any irradiance and would distort the resulting grid. To avoid this, I shift the measuring position of each spatial receiver, which is inside of a geometry. Detection of incorrectly placed spatial receivers has been done using ray-casting. In my implementation, I used Intel Embree CPU raytracer.

4.1 Usage of Irradiance Grid in Fragments

For dynamic objects, the computation of irradiance is done per fragment. In a fragment shader, every fragment determines its position in the irradiance grid. Because a normal vector of the fragment is known, it is possible to estimate, which directions of spatial receivers would affect this fragment the most. Using a dot product I measure the weight for every spatial receiver direction. I use these weights for a weighted sum of irradiance from the irradiance grid (see Figure 4).

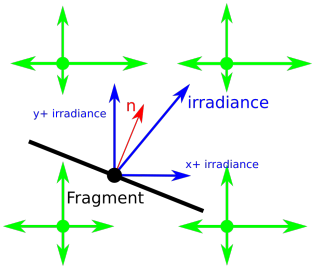

Figure 4: Visualization of irradiance reconstruction in a fragment: spatial receivers (green dots), measured directional irradiance in spatial receivers (green arrows), the fragment (black dot) with a normal vector (red arrow) and interpolated irradiances (blue arrows), summarized irradiance marked with an ”irradiance” label (a light vector).

The light vector describes from which direction comes the maximal valid irradiance to a certain fragment. Notice how this method gives us an approximation of the light vector yet just irradiance was transferred. This allows usage of normal mapping and specular reflections for dynamics objects. Furthermore, we can use the same process to apply normal mapping and specular reflections to the static scene as well. This approximation is not as accurate as transferring radiance vectors but it costs almost no additional performance and results are visually pleasing (as it will be shown in sec. 6.2).

5 Dynamic Level of Detail

My second extension uses a dynamic Level of Detail to omit computations and improve performance. The main rendering loop is separated into three stages: computation of SH coefficients for probes (λ_{ij}), computation of cluster basis vectors and computation of irradiance for every receiver using cluster basis vectors. My method is affecting every stage of this computation, so I will describe them sequentially. Every aspect of Level of Detail (LOD) such as distances, number of coefficients, number of relight rays can be freely changed in runtime.

For every level of detail, I define a distance where this level is triggered. The distance is measured between an object and a center of the camera and then compared with defined LOD distances.

5.1 Dynamic LOD for Probes

Firstly I determine to which LOD certain probe belongs. I measure this via distance between the camera center and a probe position in space.

With a known level of detail for a certain probe, it is possible to reduce calculation time in many ways. The first place where I reduced calculation time is in the number of SH coefficients.The more coefficients we use, the more accurate will be the reconstruction of the signal. So let's say we have SH of degree 4 (which is 25 coefficients, 25 basis functions to reconstruct function $f(\omega)$). Now we use SH of degree 7 (which is 64 coefficients). Even if we use SH of degree 7, those 25 coefficients from 64 in total will affect exactly the same basis functions as before. I used this property in LOD. I calculate only that number of coefficients for each probe, which is necessary for correct irradiance reconstruction. For example, in level 0, I use 64 coefficients, in level 1, I use 25 coefficients and so on. Of course, this number of coefficients must match with the computation of cluster basis vectors.

The second major improvement was made in the number of relight rays. Relight rays are used to approximate λ_{ij} coefficients using Monte-Carlo integration. The more relight rays we use, the more accurate SH coefficients are and thus the reconstruction of irradiance is more accurate. But to save computation time, it is desirable to lower the amount of these rays. In my implementation, I used 512 for level 0, 256 for level 1 and so on. With these extensions, we can notice some problems. The first problem is that a relight ray hit doesn't have to be in the same LOD as the probe is. This problem is visualized in Figure 5.

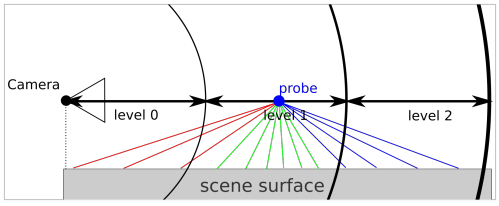

Figure 5: Visualization of different LOD for a probe and its relight rays: red rays require more detail, green rays are at same LOD and blue rays need less detail.

To solve this problem, every relight ray needs to look into the lightmap texture to correct LOD. There are two possible solutions on how to solve this. The first solution is that every relight ray remembers its 3D position of a hit and it decides itself to which LOD to look for the irradiance. The second solution is that relight rays always take irradiance from the lightmap texture with the highest LOD. The lightmap with the highest LOD is refreshed every frame and contains the newest information. In the second solution, reconstruction of the signal is less accurate but much faster to compute. I made both solutions in my implementation.

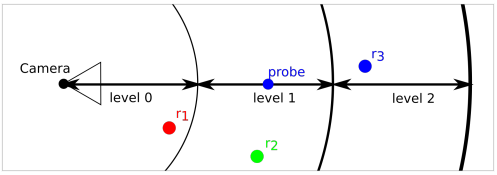

Figure 6: Visualization of different LOD for a probe and receivers supported by this probe.

Another problem comes from receivers. Even though the probe is in level 1, we cannot assign it to level 1 because there is a receiver r_1 which is in level 0 (see Figure 6).

However, only receivers which fall under the support of a certain probe are affected by it. The support of every probe was determined by a constant radius, thus the solution is to shift LOD of the probe by the constant radius. This way every receiver which needs more detailed information will get it (see Figure 7).

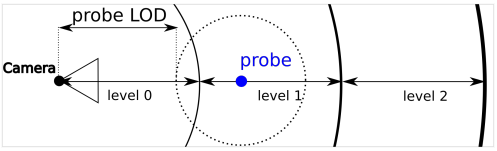

Figure 7: Visualization of a probe's LOD shifted by the probe radius.

To further improve performance I changed LOD for every probe (and its relight rays) with respect to a camera direction of view. I split surrounding of the camera into separated zones as is shown in Figure 8. To maintain good performance, I implemented this separation using a dot product with the camera view direction. This technique has been used for cluster basis vectors and irradiance receivers as well.

Note that it is important to calculate even points which are not directly visible to the viewer. Due to global illumination, even not visible points affect the final image.

5.2 Dynamic LOD for Clusters

In the beginning, we have to determine to which LOD certain cluster belongs. In the precomputation part, clusters were made by a kD-tree division. Thus, every cluster fits nicely into an AABB. I measured the closest distance from the camera center to this bounding volume and used the separated zones around the camera.

With determined LOD of the cluster, I can determine how many coefficients from probes to use for the basis

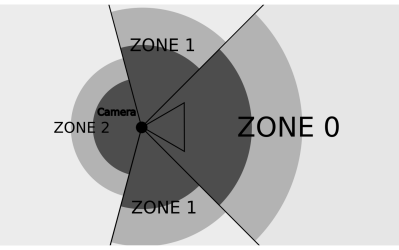

Figure 8: Visualization of separated zones around the camera, each zone has different LOD distances.

vector. In the precomputation part, two PCA matrices were calculated: matrix ΣV^t and matrix U. In this stage, we compute basis vector b, for which only matrix ΣV^t is needed.

An important property of SVD is that it maintains the order of an input vector. Thus, the order of columns in matrix ΣV^t is equal to the order of columns in the original transformation matrix (let's call that matrix A). In the algorithm, our input vector is a vector containing λ_{ij} SH coefficients of probes supporting this cluster. Because columns are in the correct order even after decomposition, we can use only those columns from ΣV^t, which we need. An example is shown in Figure 9.

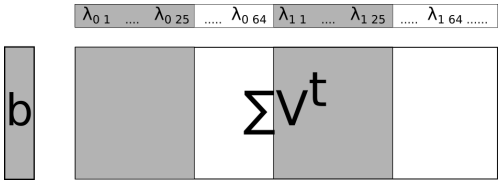

Figure 9: Visualization of partial matrix multiplication to obtain the cluster basis vector b: grey parts are used/computed, white parts are omitted.

5.3 Dynamic LOD for Receivers

In this part of the computation, irradiance is computed for every receiver, thus even for spatial receivers. In the beginning, every receiver is tested whether it has to be computed. If the receiver passes the test, irradiance is computed. The computation is omitted otherwise. The test is different if the receiver is a standard texel of lightmap texture or a spatial receiver.

If it is a standard receiver, the test starts with reading the LOD of its cluster from a texture. Then, a level of this receiver needs to be known, for obtaining the level of receiver we follow the same rules as for standard mipmapping. Thus, let's say that the coordinates of the receiver in the lightmap texture (in pixels) are (x,y). If the last bit is 0 for both coordinates, then this receiver is at least level 1. If the last 2 bits are 0, then this receiver is at least level 2 and so on. Otherwise, the receiver is level 0.

The irradiance of the receiver has to be propagated from the LOD of its cluster to the level of the receiver. The propagation is required due to possible different LOD of the cluster and level of some receiver in that cluster. So it may happen, that during rendering we require lesser detail from the cluster with higher LOD. The problem is visualized in Figure 10.

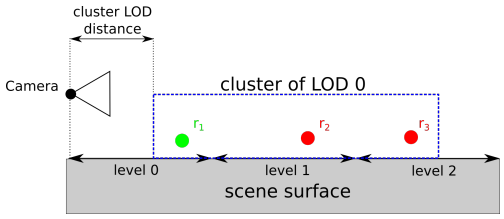

Figure 10: Visualization of the problem with receivers being in a different LOD than their cluster: receivers r_2 and r_3 are in LOD 1 and 2 respectively, but the cluster is in LOD 0.

If a tested receiver is a spatial receiver, then we check whether its voxel is active (if the voxel has been triggered by some dynamic object).

During rendering, LOD of every fragment is determined the same way as for probes. The fragment then takes irradiance from the lightmap at its LOD. To avoid sharp edges, I smooth irradiance in fragments between LODs of the lightmap texture. Furthermore, to avoid flickering due to the convergence of the system, I shift the fragment-to-camera distance by a static offset.

6 Results

Testing hardware for this section was the following PC setup: 2× Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz, RAM: 64GB, GPU NVIDIA GeForce GTX Titan Black 6GB, 980 Mhz. Renders were captured in a resolution 1280×960.

6.1 Shading of Dynamic Objects

Figure 11 demonstrates real-time color bleeding onto a dynamic container in the middle, it also shows support of normal mapping and specular reflections.

My method is also capable to cast shadows from dynamic objects using shadow maps. An example is shown in Figure 12. As it was mentioned in chapter 4, this method can also approximate normal mapping and specular reflections for a static scene without costly radiance transport. Examples are shown in Figure 13.

Quality of the shading depends on the density of the grid. To show how the density can affect shading, I prepared several scenes with a different grid density. The test results are visible in Figure 14.

The Structural Similarity index (SSIM) comparisons prove that even a sparse grid is capable to shade dynamic

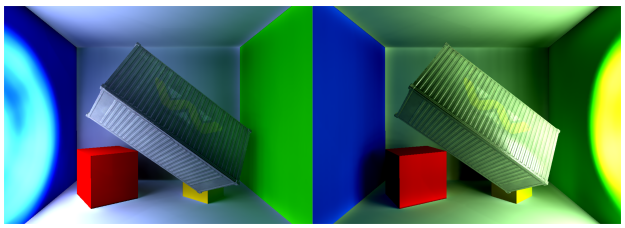

Figure 11: Example of real-time color bleeding and support of normal mapping and specular reflections for dynamic objects (model of a container).

Figure 12: Example of a shadow cast by a dynamic object (character in the middle).

objects without noticeable distortions. This gives us a great ability to cover big scenes.

6.2 Dynamic Level of Detail

An example of my Dynamic Level of Detail is shown in Figure 15. Setups used for the testing are in table 1, the distance represents how far away from the camera does this level begin. Measured results are in table 2.

For the examples, I chose high-quality setups of LOD, so the differences of rendered images are minimal. However, it is possible to tweak any property of the Dynamic LOD in run-time to increase the performance. As it was mentioned in section 5 my Dynamic LOD system is able to drastically decrease the number of computed receivers, change the drawing distance for each level of detail (Figure 16), change the degree of Spherical Harmonics for each level separately (Figure 3), change the number of relight rays and it even provides view-dependant LOD to minimize computations behind the camera.

I measured a scene with 6 different setups, data are visualized in Figure 17. The system was able to achieve speedup up to 4.34 with SSIM difference of 0.972. Quality of the image is decreasing as we omit more computation. However, the differences are still quite small. Even for the fastest tested setup, SSIM difference did not fall below 0.97.

I compared my dynamic LOD with the original method. Both methods were tested on scenes with different density of receivers. The results are shown in Figure 18.

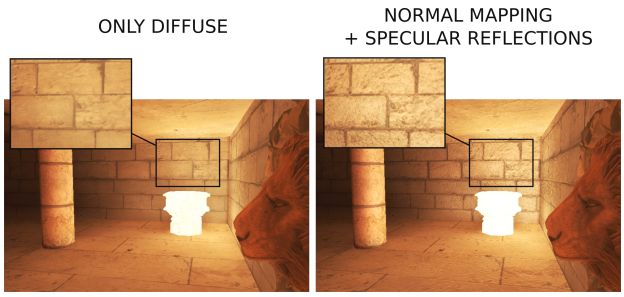

Figure 13: Example of normal mapping and specular reflections for a static scene.

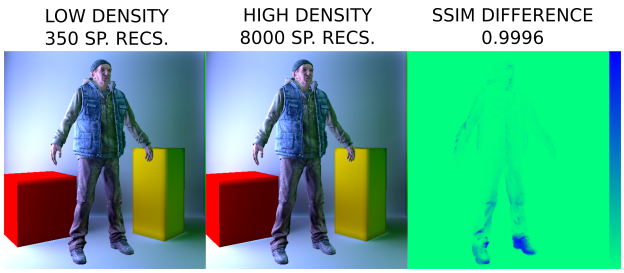

Figure 14: Testing density of the irradiance grid: low density on the left (350 spatial receivers), high density in the middle (8000 spatial receivers), SSIM difference on the right.

7 Conclusions

I reimplemented the method for real-time global illumination [4] and created two extensions. The first extension is able to shade dynamic objects in real-time with global illumination from the scene using a sparse voxel grid. Furthermore, my method is capable of normal mapping and specular reflections approximation for both dynamic objects and static scene without the need for costly radiance transport. The second extension uses a dynamic Level of Detail to omit computations and improve performance. This method achieved speedup up to 4.34 with only a slight distortion of the rendered frame.

As the method is able to cast shadows for dynamic objects only from dynamic light-sources, I want to focus on the full support of shadow casting even from indirect illumination. Another field of study would be high-frequency specular reflections, as they are not well handled by the current method.

8 Acknowledgements

This research was supported by the Czech Science Foundation under project number GA18-20374S, and the Grant Agency of the Czech Technical University in Prague, grant No. SGS19/179/OHK3/3T/13.

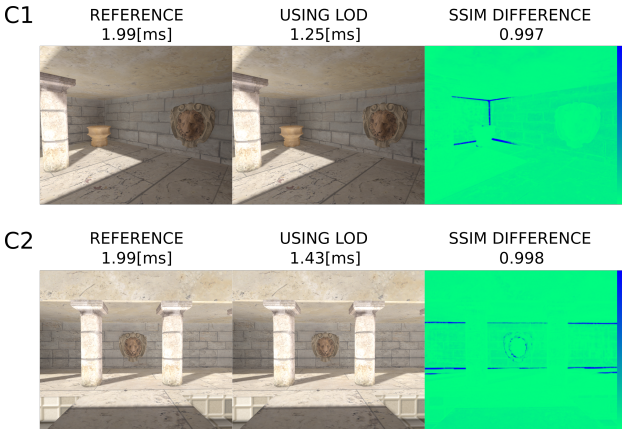

Figure 15: Dynamic LOD. The first column: a reference image without LOD. The second column: using my Dynamic LOD. The third column: SSIM difference.

	level 0	level 1	level 2	level 3
SH degree	7	6	4	3
distance	0.0	20.0	50.0	100.0

Table 1: Table of setups for LOD in Figure 15.

References

[1] John Amanatides, Andrew Woo, et al. A fast voxel traversal algorithm for ray tracing. In *Eurographics*, volume 87, pages 3–10, 1987.

[2] Robin Green. Spherical harmonic lighting: The gritty details. In *Archives of the Game Developers Conference*, volume 56, page 4, 2003.

[3] Gene Greger, Peter Shirley, Philip M Hubbard, and Donald P Greenberg. The irradiance volume. *IEEE Computer Graphics and Applications*, 18(2):32–43, 1998.

[4] Ari Silvennoinen Jaakko Lehtinen. *Real-time Global Illumination by Precomputed Local Reconstruction from Sparse Radiance Probes*. ACM Transactions on Graphics, Vol. 36, No. 6, Article 230, 2017.

[5] Morgan McGuire, Mike Mara, Derek Nowrouzezahrai, and David Luebke. Real-time global illumination using precomputed light field probes. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '17, pages 2:1–2:11, New York, NY, USA, 2017. ACM.

[6] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 527–536. ACM, 2002.

Camera	Receivers		Time [ms]		Speedup
	No LOD	LOD	No LOD	LOD	
C1	512 026	135 452	1.99	1.25	1.60
C2	512 026	217 911	1.99	1.43	1.39

Table 2: Table of results for Figure 15. Presented times are times for irradiance computation.

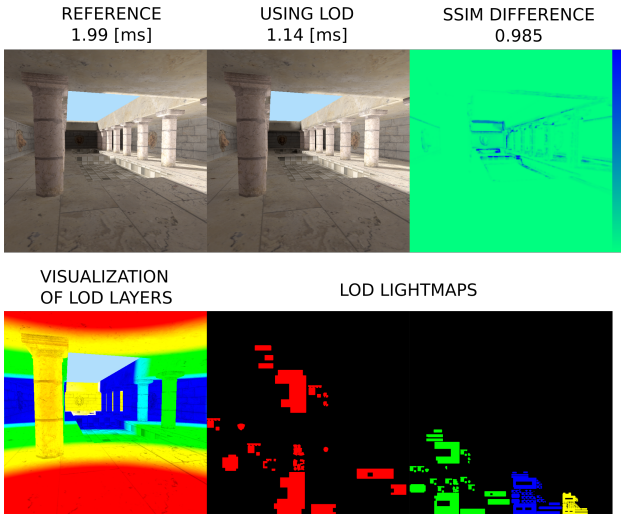

Figure 16: Dynamic LOD: change of drawing distance of each LOD layer in run-time. Notice how LOD lightmaps are only partially rendered. SSIM difference was tested against a reference image without any LOD.

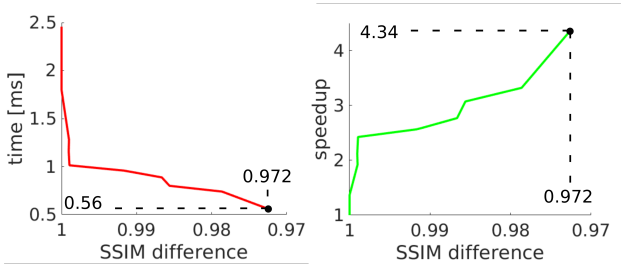

Figure 17: Comparison of different setups of Dynamic LOD. The left chart: decreasing render time, with decreasing quality of the render. The right chart: increasing speedup with decreasing quality of the render.

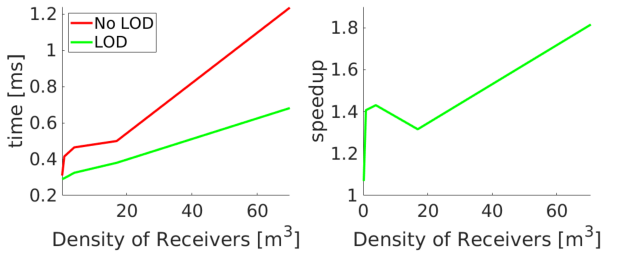

Figure 18: Comparison of No LOD and my Dynamic LOD. The left chart: LOD decreases rendering time. The right chart: speedup provided by LOD. LOD was set up to sustain SSIM difference above 0.9.